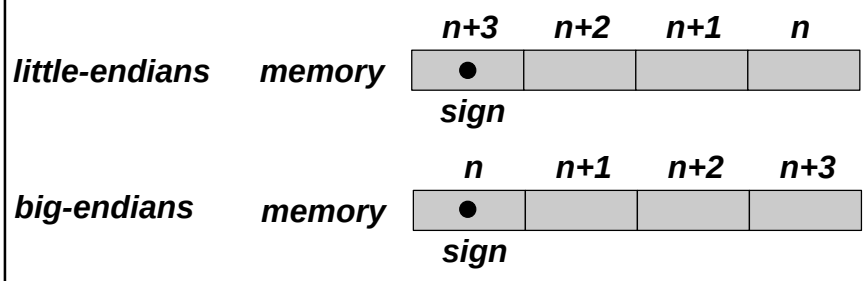
Resolving Hardware and Operating System Heterogeneity

The Problem

- Hosts of client and server might use different data representation formats. E.g. UBS:
  - Unisys Mainframe is big-endian
  - Unix servers & NT workstations are little-endians



The Problem (cont'd)

- *Different programming languages use different data representations, e.g. Character string "abc" in Pascal or C++:*

Pascal	*memory*	*3*	*a*	*b*	*c*
C++	*memory*	*a*	*b*	*c*	*\0*

Motivation

- *Data representations have to be converted between client and server*
- *Conversion should be transparent to application developer*
- *Generally achieved by middleware within presentation layer implementation*

Approaches

- *Mappings between native representations*
  - *Standardized data representation, e.g.*
    - *Sun's external data representation (XDR)*
    - *OMG's common data representation (CDR)*

    No transmission of the type definition
  - *Transmission of values and their types using an abstract syntax notation e.g*
    - *ASN.1*

OMG's Common Data Representation

- *CDR defines how ORBs of different vendors exchange data*
- *Defines how types defined in IDL are mapped to octet streams and vice-versa.*
- *Concerns mapping of*
  - *atomic types*
  - *constructed types*
  - *pseudo-types (e.g. exceptions)*
  - *object-references*

CDR Mapping of Atomic Types

- *Includes both big and little endian encodings.*
  - *Messages explicitly say which encoding was chosen.*
  - *The receiver is responsible for converting if necessary.*
  - *Reduces overhead for transfer between machines with the same representation.*
- *Mapping determines size for all atomic types (e.g 1 byte for char, 4 byte for long)*

Example of Atomic Type Mapping

- *Floating point numbers with double precision*

	Big endian		Little endian	
0	s | e1	0	f14	0
1	e2	1	f13	1
2	f1	2	f12	2
3	f2	3	f11	3
4	f3	4	f10	4
5	f4	5	f9	5
6	f5	6	f8	6
7	f6	7	f7	7
8	f7	8	f6	8
9	f8	9	f5	9
10	f9	10	f4	10
11	f10	11	f3	11
12	f11	12	f2	12
13	f12	13	f1	13
14	f13	14	e2	14
15	f14	15	s | e1	15

CDR Mapping of Constructed Types

- *Number of elements (in a given encoding)*
- *Elements as the result of mapping atomic and/or nested constructed types*
- *Example:*
  `sequence<unsigned short>:[2,3]:`

Big endian	Little endian
0	3
0	0
0	0
3	0
0	2
2	0
0	3
3	0

CDR Mapping of Object References

- *Information needed for object references:*
  - *Is it NULL (will never be used for a request)*
  - *What is the referenced object's type*
  - *Which protocols are supported*
  - *What ORB services are involved with the reference*
- *Above information provided in Interoperable Object References (IORs)*

Abstract Syntax Notation 1 (ASN.1)

- *In previous approach it was assumed*
  - *Client and server agree on message format (because stubs were created from the same interface definition)*
  - *Message format is not transmitted.*
  - *Lightweight but unreliable.*
- *ASN.1 facilitates transmission of message formats*
- *Originated in CCITT X.409. Now ISO-8824*

ASN.1 Motivation

- *Provides ability to describe information independent of way of representation*
- *Serves as a data definition language*
- *ASN.1 data definitions are converted into the ones used locally in client or server*
- *Syntax of ASN.1 similar to declaration in programming languages.*

ASN.1

- *Triple of <TYPE,LENGTH,VALUE>*
  - *TYPE can be*
    - *universally defined,*
    - *context specific,*
    - *application specific*
  - *LENGTH*
    - *Allows for transparent carrying*
    - *Undetermined lengths*
  - *VALUE*
    - *primitive (no internal structure)*
    - *constructed (with internal structure)*

ASN.1 Type Definition

```
TEAM ::= SEQUENCE {
 players    SEQUENCE {
  name      IA80String,
  club      IA80String,
  licenseNo NumericString
}
```

ASN.1 Values

```
{
 {
  name = “Juergen Klinsman”,
  license_no = “293784”
 },
 {
  name = “Alan Shearer”
  license_no = “001184”
 }
 …
}
```